

AJAX

# Lernziele

Sie wissen, was AJAX bedeutet.

Sie können AJAX im technischen Zusammenhang einordnen.

Sie kennen den grundlegenden Ablauf einer AJAX-Anfrage.

A Brain-Friendly Guide

# Head First Ajax



## Fireside Chats



Learn how to make  
your web pages  
talk and listen at  
the same time



Make your clunky  
web apps feel like  
dynamic, responsive  
desktop applications



Learn how Sally did  
two things at the same  
time with asynchronous  
programming



Transfer your  
data with plain text,  
XML, and JSON



Get a handle  
on trees and  
the Document  
Object Model

O'REILLY®

Rebecca M. Riordan

# Head First Ajax

Rebecca M. Riordan

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

# Links und Literatur

Vorschlag Literatur: Galileo Open Book (Wenz, Christian: Javascript und AJAX, Galileo Open Book)

Vorschlag Literatur: AJAX von Christian Wenz, Frankfurt 2010, Entwickler Press

Vorschlag Literatur: AJAX Head First von Rebecca Riordan

Teialehrbuch OpenBook AJAX

<http://www.teialehrbuch.de/Kostenlose-Kurse/AJAX/>

Mozilla AJAX Anleitungen, Artikel und weiteres <https://developer.mozilla.org/de/Ajax>

Webmasterpro AJAX Artikel <http://www.webmasterpro.de/coding/article/ajax-einfuehrung-uebersicht.html>

# AJAX ist nicht neu

Die aktuellsten Bücher sind von 2010.

Die Möglichkeiten von JavaScript sind heute grösser.

AJAX musste aber nicht aktualisiert werden.

# AJAX war nie neu

AJAX war nie eine neue Erfindung.

Die Techniken von AJAX existieren schon seit ca. 1998.

Zusammenfassung der Techniken erst ca. 2004.

# Was bedeutet AJAX

**A**ynchronous  
**J**avaScript  
**A**nd  
**X**ML

JavaScript



# bisher

Webseiten mit HTML5

Daten können dynamisch vom Server geladen werden  
(z. B. mit PHP)

Aktualisierung durch Reload der kompletten Seite

Während des Reloads keine Aktionen im Browser möglich

# mit AJAX

Webseiten mit HTML5

Daten können dynamisch vom Server geladen werden  
(z. B. mit PHP)

Die bestehende Seite wird aktualisiert. Kein Reload.

Während des Reloads bleibt die Seite im Browser voll funktionsfähig.

# Serverkommunikation bisher

Der User klickt auf einen Link.



Der User klickt **wieder** auf einen Link.



Der Server sendet **wieder** die komplette Seite an den Browser.

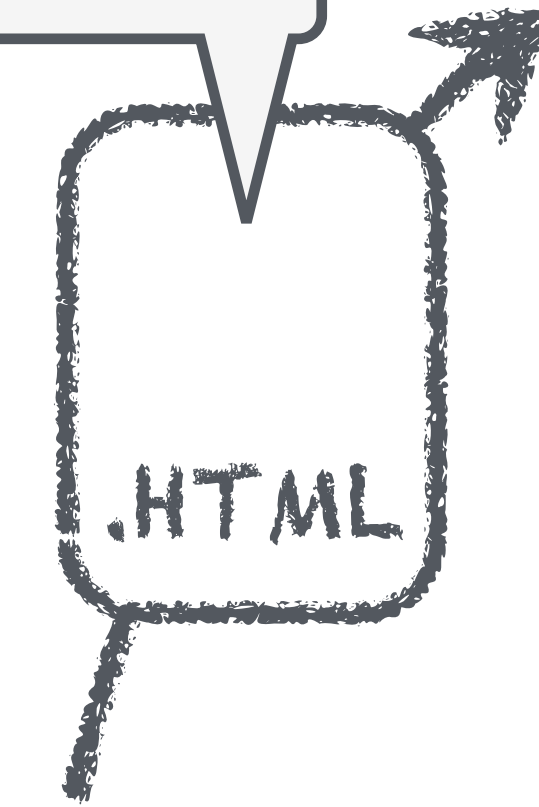
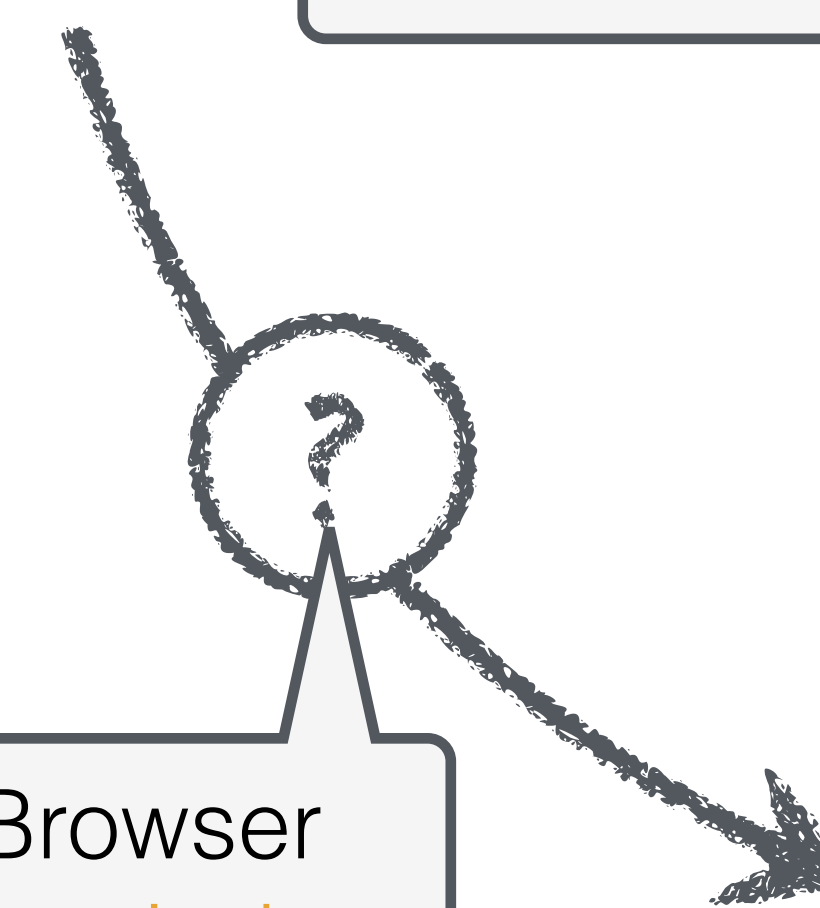
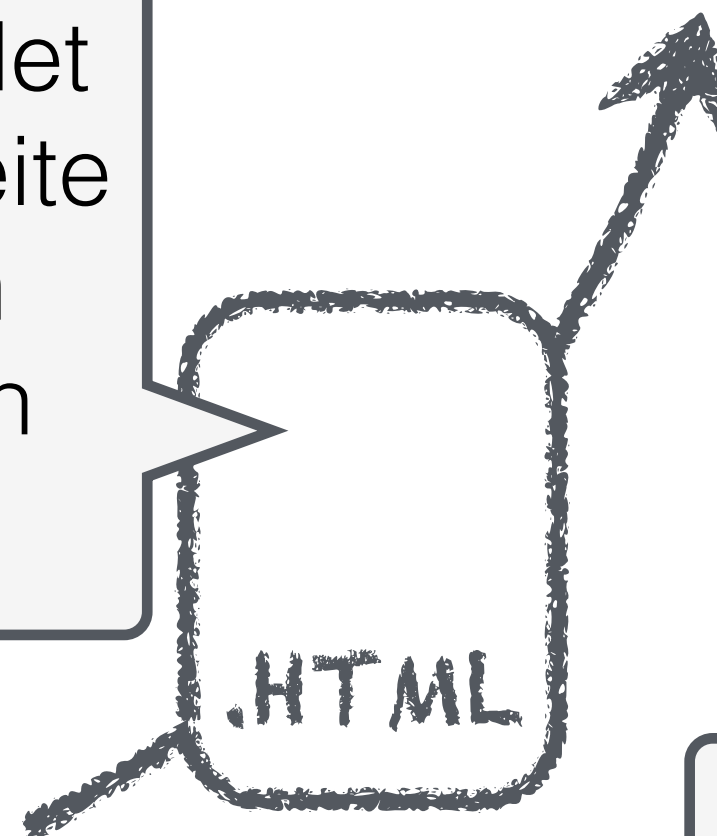
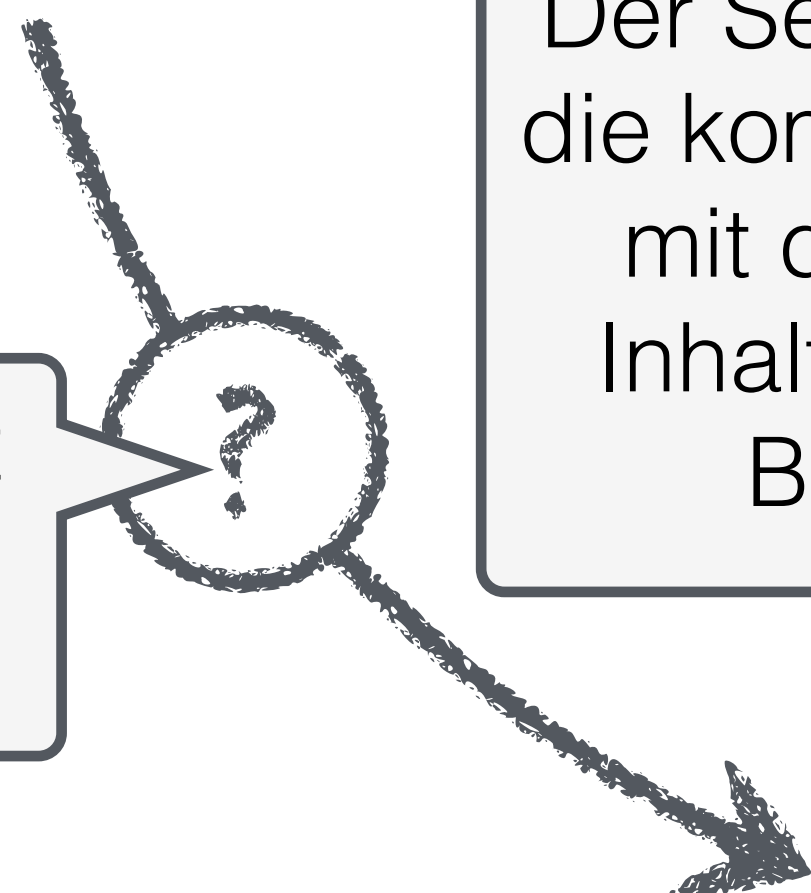
Der Server sendet die komplette Seite mit den neuen Inhalten an den Browser.

.HTML

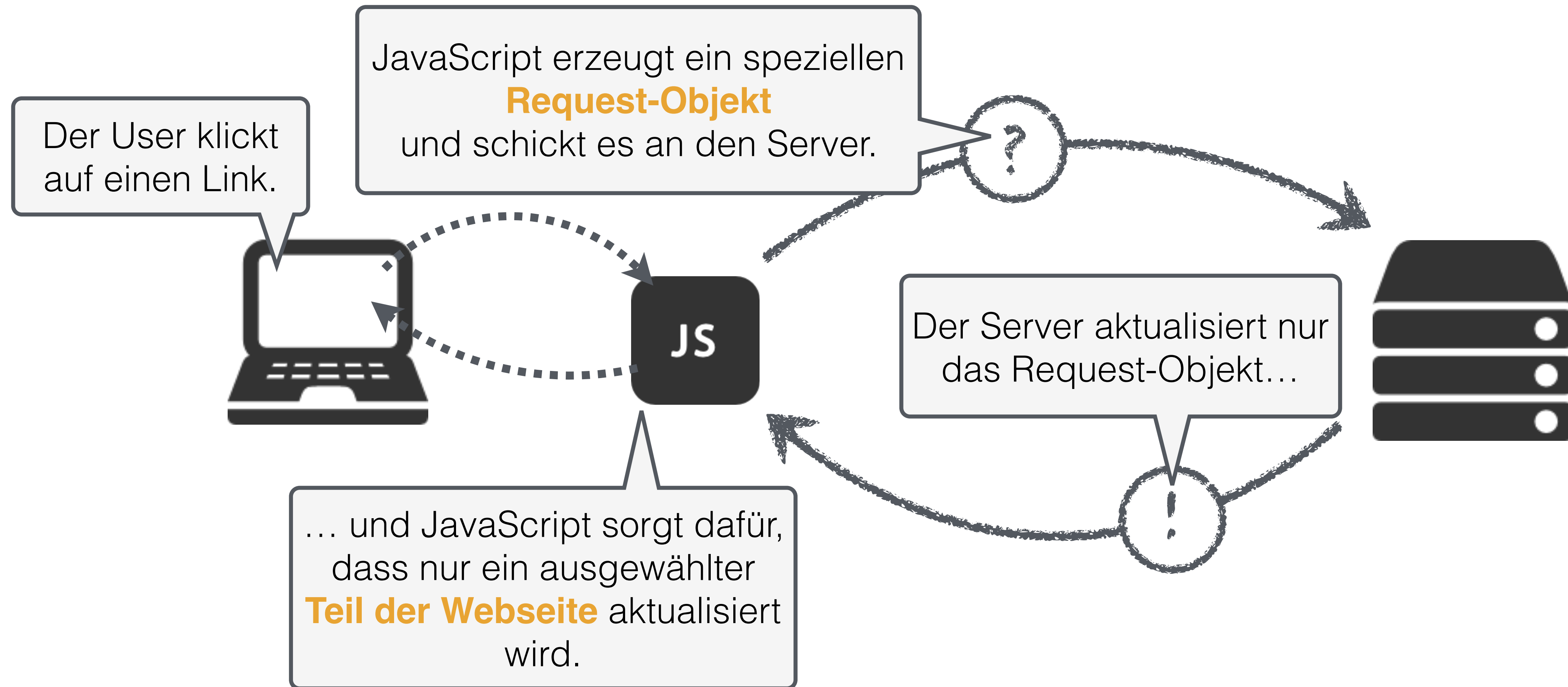
Der Browser sendet einen Request an den Server.

Der Browser sendet **wieder** einen Request.

.HTML



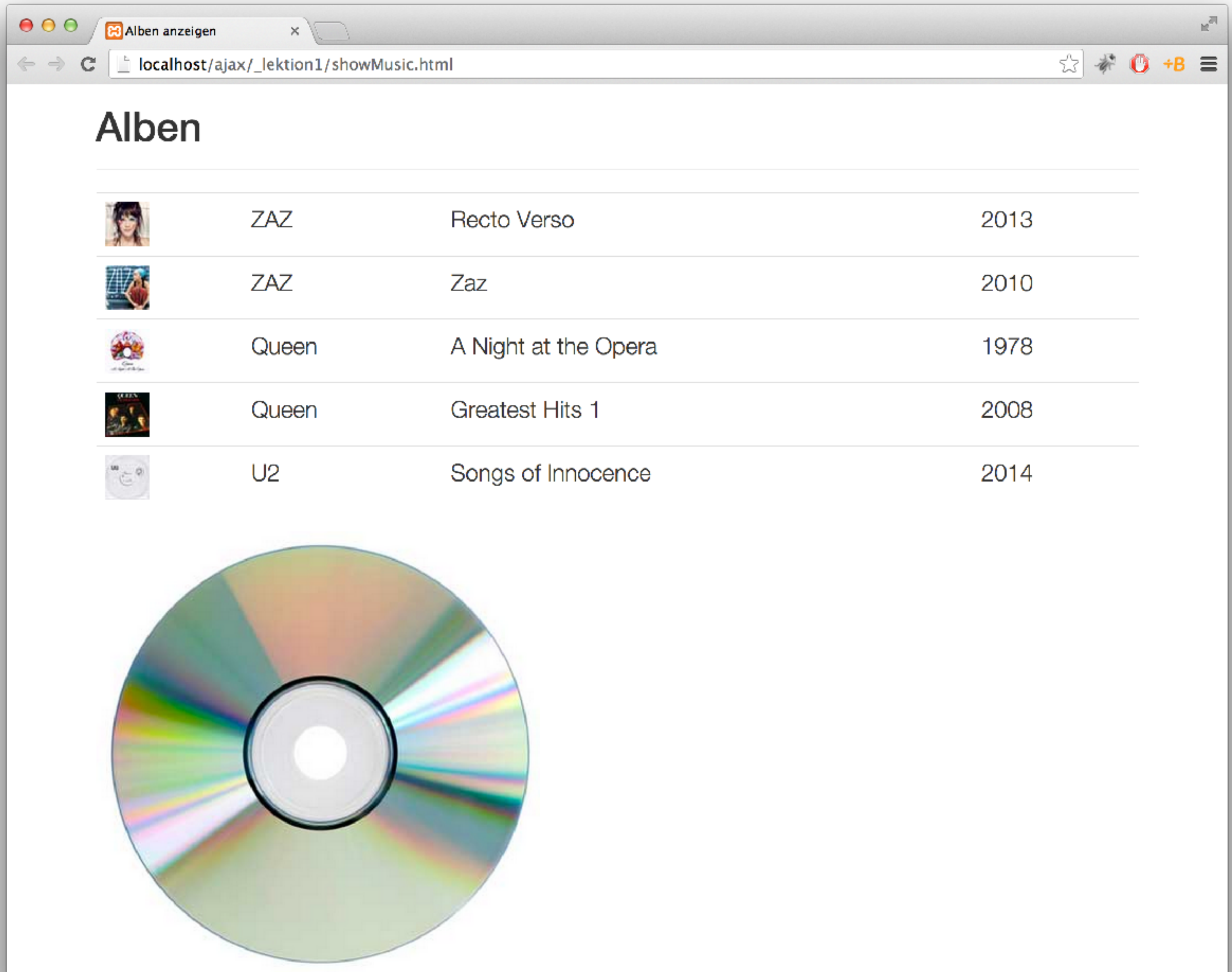
# Serverkommunikation mit AJAX



Beispiel

# Ziel

Anzeige aller Alben  
auf einer Seite.










# Ziel


Bei Klick auf ein Album wird das Cover, alle Titel mit ihrer Dauer angezeigt.

Alben anzeigen

localhost/ajax/\_lektion1/showMusic.html

## Alben

	ZAZ	Recto Verso	2013
	ZAZ	Zaz	2010
	Queen	A Night at the Opera	1978
	Queen	Greatest Hits 1	2008
	U2	Songs of Innocence	2014



	Titel	Dauer
1	The Miracle (Of Joey Ramone)	04:15
2	Every Breaking Wave	04:13
3	California (There Is No End to Love)	04:00
4	Song for Someone	03:47
5	Iris (Hold Me Close)	05:19
6	Volcano	03:14
7	Raised By Wolves	04:06
8	Cedarwood Road	04:25
9	Sleep Like a Baby Tonight	05:02
10	This Is Where You Can Reach Me Now	05:05
11	The Troubles	06:26

# ToDo

HTML vorbereiten

Seite in JavaScript initialisieren

Ein Request-Objekt erstellen

Die Details vom Server abrufen

Die Details auf der Seite anzeigen

```
<!doctype html>
<html>
<head>
...
<script src="scripts/details.js" type="text/javascript"></script>
</head>

<body>
...
  <table id="albumTable" class="table table-hover lead">
    ...
  </table>
  <div id="detailsPane" class="row">
    ...
  </div>
...
</body>
</html>
```



Verknüpfung zur JavaScriptdatei **details.js**. Sie regelt die Kommunikation mit dem Server und die Aktualisierung der HTML-Datei.

```
<!doctype html>  
<html>  
<head>
```

```
...  
<script src="scripts/details.js" type="text/javascript"></script>  
</head>
```

```
<body>
```

```
...  
  <table id="albumTable" class="table table-hover lead">  
    ...  
  </table>  
  <div id="detailsPane" class="row">  
    ...  
  </div>
```

```
...  
</body>  
</html>
```

```
<!doctype html>
<html>
<head>
```

```
...
<script src="scripts/details.js" type="text/javascript"></script>
</head>
```

```
<body>
```

```
...
  <table id="albumTable" class="table table-hover lead">
```

```
    ...
  </table>
```

```
  <div id="detailsPane" class="row">
```

```
    ...
  </div>
```

```
...
</body>
</html>
```

Die Tabelle dient der Auswahl des Albums. Durch Klick auf eine Zeile wird ein **request** ausgelöst, der neue Daten lädt.

```
<!doctype html>
<html>
<head>
...
<script src="scripts/details.js" type="text/javascript"></script>
</head>

<body>
...
  <table id="albumTable" class="table table-hover lead">
    ...
  </table>
  <div id="detailsPane" class="row">
    ...
  </div>
...
</body>
</html>
```

Im div-Element mit der id detailsPane wird nach der Aktualisierung das Ergebnis angezeigt

```
<table id="albumTable" class="table table-hover lead">
  <tr title="rectoVerso" id="album1">
    ...
  </tr>
  <tr title="zaz" id="album2">
    ...
  </tr>
  <tr title="nightAtTheOpera" id="album3">
    ...
  </tr>
  <tr title="qgh1" id="album4">
    ...
  </tr>
  <tr title="u2a2014" id="album5">
    ...
  </tr>
</table>
```

Die Werte des id-Attributs der Table, sowie die der id- und title-Attribute der Zeilen werden später von JavaScript benötigt.

Es werden nur innerhalb **detailsPane**  
Änderungen durch JavaScript vorgenommen

```
<div id="detailsPane" class="row">  
  <div class="col-md-6">  
      
  </div>  
  <div id="description" class="col-md-6">  
  
  </div>  
</div>
```

Im **img**-Element mit der id **cover**  
wird das Bild durch das gewählte  
Albumcover ersetzt.

In **description** wird eine Tabelle mit den  
Titeln des gewählten Albums geschrieben.

# ToDo

~~HTML vorbereiten~~

Seite in JavaScript initialisieren

Ein Request-Objekt erstellen

Die Details vom Server abrufen

Die Details auf der Seite anzeigen



details.js

```
window.onload = initPage;
```

```
function initPage() {  
    albumList = document.getElementById("albumTable").getElementsByTagName("tr");  
  
    for (var i = 0; i < albumList.length; i++) {  
        album = albumList[i];  
  
        album.onclick = function() {  
            detailURL = 'cover/' + this.title + '.jpg';  
            document.getElementById("cover").src = detailURL;  
  
            getDetails(this.id);  
        }  
    }  
}  
...  
}
```



details.js

```
window.onload = initPage;
```

```
function initPage() {  
    albumList = document.getElementById("albumTable").getElementsByTagName("tr");  
  
    for (var i = 0; i < albumList.length; i++) {  
        album = albumList[i];  
  
        album.onclick = function() {  
            detailURL = 'cover/' + this.title + '.jpg';  
            document.getElementById("cover").src = detailURL;  
  
            getDetails(this.id);  
        }  
    }  
}  
...
```



```
window.onload = initPage;
```

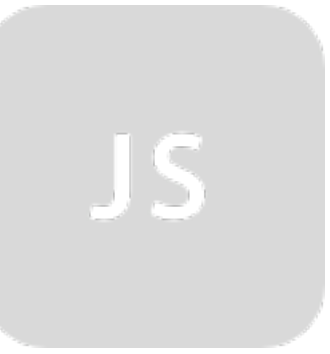
```
function initPage() {  
    albumList = document.getElementById("albumTable").getElementsByTagName("tr");  
  
    for (var i = 0; i < albumList.length; i++) {  
        album = albumList[i];  
  
        album.onclick = function() {  
            detailURL = 'cover/' + this.title + '.jpg';  
            document.getElementById("cover").src = detailURL;  
  
            getDetails(this.id);  
        }  
    }  
}  
...  
...  
...
```

Array mit allen Zeilen (**tr**) der  
Tabelle **albumTable**

```
window.onload = initPage;
```

```
function initPage() {  
    albumList = document.getElementById("albumTable").getElementsByTagName("tr");  
  
    for (var i = 0; i < albumList.length; i++) {  
        album = albumList[i];  
  
        album.onclick = function() {  
            detailURL = 'cover/' + this.title + '.jpg';  
            document.getElementById("cover").src = detailURL;  
  
            getDetails(this.id);  
        }  
    }  
}  
...  
...
```

Eine Schleife für alle Elemente  
aus dem Array **albumList**



details.js

```
window.onload = initPage;
```

```
function initPage() {  
    albumList = document.getElementById("albumTable").getElementsByTagName("tr");  
  
    for (var i = 0; i < albumList.length; i++) {  
        album = albumList[i];  
  
        album.onclick = function() {  
            + this.title + '.jpg';  
            yId("cover").src = detailURL;  
        }  
    }  
}  
...
```

Bei jedem Schleifendurchlauf  
wird der eine Tabellenzeile in die  
Variable **album** geschrieben...



details.js

```
window.onload = initPage;
```

```
function initPage() {  
    albumList = document.getElementById("albumTable").getElementsByTagName("tr");  
  
    for (var i = 0; i < albumList.length; i++) {  
        album = albumList[i];  
  
        album.onclick = function() {  
            detailURL = 'cover/' + this.title + '.jpg';  
            document.getElementById("cover").src = detailURL;  
            getDetail();  
        }  
    }  
}
```

... und für das onClickEvent eine anonyme Funktion definiert.

```
...  
...  
...
```



details.js

```
window.onload = initPage;
```

```
function initPage() {
```

Beim Funktionsaufruf wird zuerst der Pfad für das Albumcover definiert und in die Variable **detailURL** geschrieben. In **this.title** steht der Wert des **title**-Attributs der aktuellen Tabellenzeile. Er wurde in der HTML-Datei definiert.

```
getElementsByTagName("tr");
```

```
    album.onclick = function() {  
        detailURL = 'cover/' + this.title + '.jpg';  
        document.getElementById("cover").src = detailURL;
```

Dann wird der Wert des src-Attributs des Elements mit der id cover (das Bild) durch den Inhalt der Variable ersetzt.

```
    }
```

```
}
```

```
}
```

```
...
```



details.js

```
window.onload = initPage;
```

```
function initPage() {  
    albumList = document.getElementById("albumTable").getElementsByTagName("tr");  
  
    for (var i = 0; i < albumList.length; i++) {  
        album = albumList[i];  
  
        album.onclick = function() {  
            detailURL = 'cover/' + this.title + '.jpg';  
            document.getElementById("cover").src = detailURL;  
  
            getDetails(this.id);  
        }  
    }  
}
```

Zuletzt wird die **getDetails**-Funktion aufgerufen und als Parameter der Inhalt des **id**-Attributs aus dem gewählten **tr**-Elements übergeben.

```
...
```



details.js

```
window.onload = initPage;
```

```
function initPage() {  
    albumList = document.getElementById("albumTable").getElementsByTagName("tr");  
  
    for (var i = 0; i < albumList.length; i++) {  
        album = albumList[i];  
  
        album.onclick = function() {  
            detailURL = 'cover/' + this.title + '.jpg';  
            document.getElementById("cover").src = detailURL;  
  
            getDetails(this.id);  
        }  
    }  
}  
...  
}
```

# ToDo

~~HTML vorbereiten~~

~~Seite in JavaScript initialisieren~~

Ein Request-Objekt erstellen

Die Details vom Server abrufen

Die Details auf der Seite anzeigen



Ein Request-Objekt steuert die Verbindung vom dem Browser zu einer angeforderten Datei.

```
function createRequest() {
  try {
    // moderne Browser
    request = new XMLHttpRequest();
  } catch (tryMS) {
    try {
      // MS Internet Explorer (ab v6)
      request = new ActiveXObject(„Microsoft.XMLHTTP“);
    } catch (otherMS) {
      try {
        // MS Internet Explorer (ab v5)
        request = new ActiveXObject("Msxml2.XMLHTTP");
      } catch (failed) {
        request = null;
      }
    }
  }
  return request;
}
```

```
function createRequest() {  
  try {  
    // moderne Browser  
    request = new XMLHttpRequest();  
  } catch (tryMS) {  
    try {  
      // MS Internet Explorer (ab v6)  
      request = new ActiveXObject(„Microsoft.XMLHTTP“);  
    } catch (otherMS) {  
      try {  
        // MS Internet Explorer (ab v5)  
        request = new ActiveXObject("Msxml2.XMLHTTP");  
      } catch (failed) {  
        request = null;  
      }  
    }  
  }  
  return request;  
}
```

Der erste Versuch

Sollte der erste Versuch fehlschlagen, wird der Fehler hier abgefangen...

... und direkt der zweite Versuch gestartet.

Wenn auch der zweite Versuch fehlschlägt, wird der Fehler hier abgefangen...

... und direkt der dritte Versuch gestartet.

Wenn auch der dritte Versuch scheitert, wird dieser Codeblock ausgeführt.

```

function createRequest() {
    try {
        // moderne Browser
        request = new XMLHttpRequest();
    } catch (tryMS) {
        try {
            // MS Internet Explorer (ab v6)
            request = new ActiveXObject("Microsoft.XMLHTTP");
        } catch (otherMS) {
            try {
                // MS Internet Explorer (ab v5)
                request = new ActiveXObject("Msxml2.XMLHTTP");
            } catch (failed) {
                request = null;
            }
        }
    }
    return request;
}

```

Wir versuchen mit JavaScript ein Request-Objekt zu erzeugen und in die Variable **request** zu schreiben. Leider ist die Erstellung eines Request-Objekts von Browser zu Browser unterschiedlich. Daher müssen wir mehrere Versuche starten, um alle Browser abzudecken.

Wenn keiner der drei Versuche zu einem Request-Objekt führt, setzen wir die Variable auf **null**.

Zuletzt geben wir den Inhalt der Variable zurück.

# ToDo

~~HTML vorbereiten~~

~~Seite in JavaScript initialisieren~~

~~Ein Request Objekt erstellen~~

Die Details vom Server abrufen

Die Details auf der Seite anzeigen

```
function getDetails(itemName) {  
    request = createRequest();  
  
    if (request == null) {  
        alert("Es konnte keine Verbindung hergestellt werden.");  
        return;  
    }  
  
    var url= "getDetails.php?id=" + escape(itemName);  
    request.open("GET", url, true);  
    request.onreadystatechange = displayDetails;  
    request.send(null);  
}
```

Die **getDetails**-Funktion wird durch Klick auf eine Tabellenzeile ausgelöst. Der übergebene Parameter ist der Wert aus dem **id**-Attribut der entsprechenden Tabellenzeile.

```
function getDetails(itemName) {  
    request = createRequest();
```

Wir rufen die eben erstellte Funktion auf und schreiben den Rückgabewert in die Variable **request**.

```
    if (request == null) {  
        alert("Es konnte keine Verbindung hergestellt werden.");  
        return;  
    }
```

Wenn kein Request-Objekt erzeugt werden konnte, wird eine Fehlermeldung angezeigt und die Funktion durch return beendet.

```
    var url= "getDetails.php?id=" + escape(itemName);  
    request.open("GET", url, true);  
    request.onreadystatechange = displayDetails;  
    request.send(null);  
}
```

```
function getDetails(itemName) {  
    request = createRequest();
```

Das Skript in der externen Datei **getDetails.php** gibt die angeforderten Daten zurück.  
Der Funktionsparameter wird als **id** der URL mit **?** angehängt.  
Die **escape** transcodiert etwaige Sonderzeichen wie \* @ - \_ + . / oder das Leerzeichen.

```
var url= "getDetails.php?id=" + escape(itemName);  
request.open("GET", url, true);
```

```
request.onreadystatechange = displayDetails;
```

**open** ist eine Funktion des Request-Objekts. Sie steuert wie die Verbindung zustande kommen soll.  
Die drei Parameter definieren die Verbindungsmethode (**POST/GET**), die Adresse (die wir in der Variable **url** gespeichert haben) der Datei und die Verbindungsart (asynchron=**true**, synchron=**false**)

```
}
```



```
function getDetails(itemName) {  
    request = createRequest();
```

```
    if (request == null) {
```

**onreadystatechange** ist ebenso eine Funktion des Request-Objekts. Sie wird während der verschiedenen Phasen der externen Datenverarbeitung abgefeuert. Bei jeder Phase wird die Funktion **displayDetails** aufgerufen. Dort fragen wir ab, in welcher Ladephase wir uns befinden.

```
    var url= "getDetails.php?id=" + escape(itemName);  
    request.open("GET", url, true);
```

```
    request.onreadystatechange = displayDetails;  
    request.send(null);
```

```
}
```

**send** schickt schliesslich den Request ab. Ein spezieller Parameter wird nur benötigt, wenn wir mit der Methode **POST** senden.

# ToDo

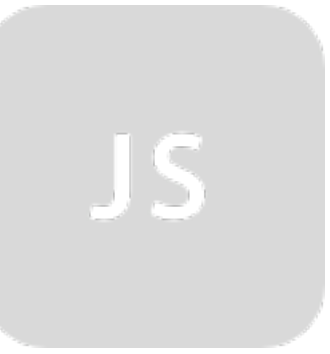
~~HTML vorbereiten~~

~~Seite in JavaScript initialisieren~~

~~Ein Request Objekt erstellen~~

~~Die Details vom Server abrufen~~

Die Details auf der Seite anzeigen



details.js

```
function displayDetails() {  
    if (request.readyState == 4) {  
        if (request.status == 200) {  
            detailDiv = document.getElementById("description");  
            detailDiv.innerHTML = request.responseText;  
        }  
    }  
}
```

`displayDetails` wird von `request.onreadystatechange` bei jedem Phasenwechsel aufgerufen. Die einzelnen Phasen (`readyState`) werden durch Nummern angegeben:

- `0`: request ist noch nicht initialisiert
- `1`: Server-Verbindung steht
- `2`: request wurde vom Server empfangen
- `3`: request wird bearbeitet
- `4`: request ist abgeschlossen und übertragen

```
function displayDetails() {  
    if (request.readyState == 4) {  
        if (request.status == 200) {
```

```
            detailDiv = document.getElementById("description");
```

Ebenso gibt das request-Objekt das `status`-Attribut zurück. Folgende Werte sind möglich:

- `200`: "OK"
- `404`: Seite nicht gefunden.

```
        }  
    }  
}
```

Das Element mit der `id="description"` wird in die Variable `detailDiv` geschrieben.

```
function displayDetails() {  
  if (request.readyState == 4) {  
    if (request.status == 200) {  
      detailDiv = document.getElementById("description");  
      detailDiv.innerHTML = request.responseText;  
    }  
  }  
}
```

`responseText` ist der vom `request` aus dem externen Skript ausgelesene Text und ein weiteres Attribut des Request-Objekts. Er wird in das zuvor definierte Element geschrieben.